**Looting Looters**

**Technical Design Document**

Version 0.04

**Version History**

|  |  |  |
| --- | --- | --- |
| **Version Number** | **Edited By** | **Date** |
| 0.01 | Travis Britton & Dayton Heywood | 20/01/2019 |
| 0.02 | Travis Britton | 21/02/2019 |
| 0.03 | Dayton Heywood | 28/02/2019 |
| 0.04 | Travis Britton | 01/03/2019 |
|  |  |  |
|  |  |  |
|  |  |  |

**Table of Contents**

[**Game Overview**](#_wo8flfdtjnsy) **3**

[Game Summary](#_30j0zll) 3

[Platform](#_1fob9te) 3

[**Development Overview**](#_3znysh7) **3**

[Development Team](#_2et92p0) 3

[Development Environment](#_tyjcwt) 3

[*Development Hardware*](#_3dy6vkm) *3*

[*Development Software*](#_1t3h5sf) *3*

[*External Code*](#_4d34og8) *3*

[**Game Mechanics**](#_2s8eyo1) **4**

[Main Technical Requirements](#_17dp8vu) 4

[Architecture](#_3rdcrjn) 4

[Game Flow](#_26in1rg) 4

[Graphics](#_lnxbz9) 4

[Audio](#_35nkun2) 4

[Artificial Intelligence](#_ybxg3l4lydjh) 5

[Networking](#_44sinio) 5

[Physics](#_2jxsxqh) 5

[Game Objects and Logic](#_z337ya) 5

[Data Management and Flow](#_3j2qqm3) 7

[**User Interface**](#_1y810tw) **7**

[Game Shell](#_4i7ojhp) 7

[Play Screen](#_2xcytpi) 7

[**Technical Risk**](#_7ghib4ef3nlm) **7**

# 

# Game Overview

## Game Summary

Looting Looters is a competitive first-person online multiplayer game where the goal is to collect as much loot as possible without getting caught by the castle guard. The winner of a round is determined by whoever scored the most points, not who got caught last... so be aggressive, be smart and LOOT! The players will have access to different traps which can be used to affect other players or the guards as well.

## Platform

PC

# Development Overview

## Development Team

Travis Britton - Programmer

Dayton Heywood - Programmer

Valerie Aubut-McWhirter - Artist

Raffaele Gambuto - Artist

## Development Environment

### Development Hardware

Different environments were used. Here are some baselines.

OS: Windows 10 Home & Pro

CPU: AMD Ryzen 1800x, Intel i7-8750h

GPU: GTX 1050Ti, GTX 1070

RAM: 8GB+ DDR4

### Development Software

Visual Studio 2017

Unreal Engine 4.20

3DS Max

Photoshop

Illustrator

Git & Github

Google Docs

Dia

Notepad++

### External Code

Unreal Engine Code

Apex Destruction plugin for Unreal Engine

UMG Library for Unreal Engine

# Game Mechanics

## Main Technical Requirements

* Unreal C++ project.
* AI (alpha 1), networking, and match-making elements(alpha 2).
* Randomly generate level as a collection of rooms
* Populate rooms with randomly selected furniture assets
* Generate room connections via doors randomly connected which can be altered during runtime
* Create traps that can be placed by characters and affect gameplay
* The ability to gather loot which gives score and possibly traps
* Interactable objects which the player can pickup, rotate, drop, throw, etc. When thrown if it impacts something it will destruct (explode into pieces) and then despawn.

## Architecture

See the UML folder for the UML. It does not properly fit here.

See the Pseudocode folder for the Pseudocode.

LEVEL ARCHITECTURE:

Upon game start, the Game randomly generates a level using room blueprints, asset blueprints and loot blueprints. No single runtime of the game will be the same (very unlikely). The level structure of actors is as follow:

GameMode -> Rooms - > Assets -> Loot

The GameMode holds all Rooms, each Room holds all Assets they own, all Assets hold all Loot they own etc.

The Game mode determines when the rooms will generate loot for the player. Currently loot is just generated at the start and it auto-respawns, but as we flesh it out, we will have the GameMode call a spawning function on a Timer that will split loot spawning across the rooms in a randomized manner.

## Game Flow

Currently in Alpha 1 there is only 1 real gamestate which is gameplay. Alpha 2 will have a main menu and options screen. For gameplay there should be gameplay, paused and spectating.

## Graphics

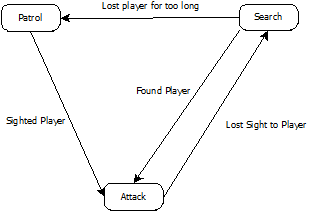
The game runs in 3D using the built in tools of Unreal engine and their constraints on content.

## Audio

No audio present in alpha 1.

## 

## Artificial Intelligence



The AI’s main objective is to patrol the map. The AI will enter a room, Find a door in the room and then try to get to it and go through it. Once they go through it this repeats in the new room.

The AI has a PawnSensingComponent with vision turned on.

If he sees the player with this component he will enter attack mode and chase the player. If he contacts the player he will “kill” the player.

If he loses sight of the player he enters search mode. In search he stores the player’s last known position and goes there to see if he can find the player. If he still can’t find the player he will try to go through the last door the player went through. If he still can’t find the player after a certain amount of time he will go back to patrol mode. If he does see the player again he re-enters attack mode.

## Networking

Not present in Alpha 1 but will be in subsequent releases.

## Physics

Physics is handled by the default methods inside of Unreal Engine 4.20. Using the basic box, capsule and sphere colliders with Unreal’s physics backend.

## Game Objects and Logic

LEVEL LOADING:

Upon game start, the Game randomly generates a level using room blueprints, asset blueprints and loot blueprints. No single runtime of the game will be the same (very unlikely to get a similar seed). The level structure of actors is as follow:

GameMode -> Rooms - > Assets -> Loot

The GameMode holds all Rooms, each Room holds all Assets they own, all Assets hold all Loot they own etc.

The Game mode determines when the rooms will generate loot for the player. Currently loot is just generated at the start and it auto-respawns, but as we flesh it out, we will have the GameMode call a spawning function on a Timer that will split loot spawning across the rooms in a randomized manner.

RANDOMIZATION AND HOW ITS DONE:

There’s numerous levels of randomization to ensure a unique seed on each run. A step by step process of the creation of a game level ensures that:

Step 1 - Generate room meshes randomly from a list of loaded blueprints

Step 2 - Connect the doors of each room together randomly (sometimes there is no connection at all)

Step 3 - Each room spawns their game assets randomly from a list of loaded blueprints using type specifiers

Step 4 - Assets spawn their loot (but they are invisible). The loot they spawn is also random.

Step 5 - On intervals during runtime, the GameMode will tell Rooms to spawn a random amount of loot.

Step 6 - Those rooms will choose assets at random and have them ‘spawn’ loot. Behind the scenes we are actually just turning loot visible.

THE PLAYERS

The players in-game will spawn in Room 1. At that point it’s up to them where they go. They can touch doors to teleport to other rooms and loot the loot they find. They can get traps as well from the loot and use them to hinder other players or the guard. If they’re touched by the guard they die and will swap to a spectator camera that will let them continue to enjoy the game from the guard’s point of view (not currently in alpha 1).

THE GUARD

The guards in-game will spawn in their own special room (not currently in alpha 1). They will travel through the Rooms in the level in an attempt to find a Player. If the guard touches a player they will die and can no longer loot anything. If the guard is chasing a player but loses them they will attempt to find that player by using the player’s last known location as well as their last door accessed before losing contact. If at that point they still can’t find the player they go back to patrolling otherwise they attack the player again. During patrol mode the guard will begin using traps as well to hinder the players using timer logic and random chance based on how long it’s been since the last time the guard dropped a trap (not currently in alpha 1). The guard has infinite traps.

LOOT

Loot is spawned by Assets. By default they are will be non interactable and invisible. When they are flagged as active they’ll be seen and can be looted by the player. The amount of score given is random (clamped to 2 values) as well as a random chance to give a trap. There are multiple tiers of Loot that will have higher chances at getting a trap or guaranteed chances at specific traps (this will likely be expanded on as we’re currently in the design phase of loot).

TRAPS

Traps are randomly given to the Player when they loot Loot. Traps are placed by both the Player and the Guard. They cannot affect the actor that placed the trap. Their effects are always negative and are used as a way to hinder guards and other players. Current traps include:

Stop Trap - The triggering actor is unable to move for a short period of time.

Slow Trap - The triggering actor’s movement speed is slowed for a period of time.

Blinding Trap - NYI

More will be added over time.

## Data Management and Flow

Due to the nature of the game being basically a random rogue-like nothing is really saved. All assets are loaded on level launch, including the random generation. Nothing gets saved upon exiting the game aside from potentially game settings.

# User Interface

This section describes how information is presented to the player, and how input is received in return. This includes display layout, feedback, controls, and so on.

## Game Shell

See Game Design Document for details.

## Play Screen

See Game Design Document for details.

# 

# Technical Risk

Non-Euclidean Level Design:

Every time the game is launched the level is made up of rooms which are not actually physically attached. Instead the “Doors” are basically just teleporters. Getting this to function reliably and with the AI guards is the biggest problem. The goal is to tackle things one at a time. First have the doors existing as actors, then have them teleport things then act as waypoints for the AI, etc.

Networking:

No one on the team has worked on networking. Through class we are learning how to properly plan and setup the project so the networking can work. It changes several aspects of design and implementation. Knowing what it changes largely resolves this issue.

Random Level Generation:

Every time the game is run the level the player experiences should be relatively different. Different orders to the rooms, different furniture and loot spawned in the rooms, different connections between the rooms, etc. This is a lot of logic to work out but tackling it piece-by-piece limits the difficulty.